

Introduction to Lambda Calculus

York University

Department of Computer Science and Engineering

Overview

- Functions
- λ -calculus : λ -notation for functions
- Free and bound variables
- α - equivalence and β -reduction
- Connection to LISP

[ref.: Chap. 1 & 2 of Selinger's lecture notes on Lambda Calculus:
<http://www.mathstat.dal.ca/~selinger/papers/lambdanotes.pdf>]

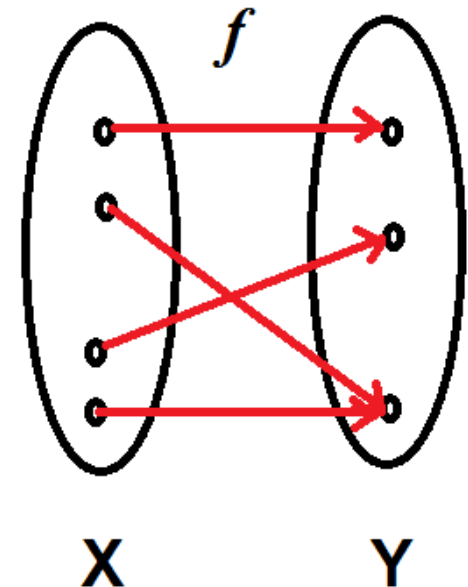
[also Wikipedia on Lambda calculus]

[I am using George Tourlakis' notations for renaming and substitution]

Extensional view of Functions

- “Functions as graphs”:
 - each function f has a fixed domain X and co-domain Y
 - a function $f : X \rightarrow Y$ is a set of pairs $f \subseteq X \times Y$ such that for each $x \in X$, there exists exactly one $y \in Y$ such that $(x, y) \in f$.
- Equality of functions:
 - Two functions are equal if given the same input they yield the same output

$$f, g : X \rightarrow Y, \quad f = g \Leftrightarrow \forall x \in X, f(x) = g(x)$$



Intensional view of Functions

- “Functions as rules”:
 - Functions defined as rules, e.g. $f(x) = x^2$
 - Not always necessary to specify domain and co-domain
- Equality of functions:
 - Two functions are equal if they are defined by (essentially) the same formula
- Comparing the two views
 - Graph model is more general, does not need a formula
 - Rule model is more interesting for computer scientists (How can it be calculated? What is the time/memory complexity? etc)

3 observations about functions

$f(x)=x$ is the identity function

$g(x)=x$ is also the identity function

→ Functions do not need to be explicitly named

→ Can be expressed as $x \mapsto x$

$(x, y) \mapsto x - y$

$(u, v) \mapsto u - v$ they are the same

→ The specific choice for argument names is irrelevant

$(x, y) \mapsto x - y$

$x \mapsto (y \mapsto x - y)$

→ Functions can be re-written in a way to accept only one single input (called **currying**)

Lambda Calculus

- These 3 observations are motivations for a new notation for functions: Lambda notation
- λ -calculus: theory of functions as formulas
- Easier manipulation of functions using expressions
- Examples of λ -notation:
 - The identity function $f(x)=x$ is denoted as $\lambda x.x$
 - $\lambda x.x$ is the same as $\lambda y.y$ (called **α -equivalence**)
 - Function f defined as $f : x \mapsto x^2$ is written as $\lambda x.x^2$
 - $f(5)$ is $(\lambda x.x^2)(5)$ and evaluates to 25 (called **β -reduction**)

More examples

- Evaluate

$$\begin{aligned} & (\lambda x.((\lambda y.x^2 + y^3)(2)))(3) \\ &= (\lambda x.x^2 + 2^3)(3) = (3^2 + 2^3) = 17 \end{aligned}$$

- Evaluate

$$\begin{aligned} & (\lambda x.(\lambda y.x^2 + y^3))(2)(3) \\ &= (\lambda y.2^2 + y^3)(3) = (2^2 + 3^3) = 31 \end{aligned}$$

Lambda terms

- λ -term calculation:
 1. A **variable** is a λ -term (for example x, y, \dots)
 2. If M is a λ -term and x is a variable, then $(\lambda x.M)$ is a λ -term (called a **lambda abstraction**)
 3. If M and N are λ -terms, then (MN) is a λ -term (called an **application**)

– Note in λ -notation we write (fx) instead of $f(x)$

Example: Write the steps in λ -term calculation of

$$(\lambda x.(\lambda y.(\lambda z.((xz)(yz))))))$$

$$x, \quad y, \quad z, \quad (xz), \quad (yz), \quad ((xz)(yz)), \quad (\lambda z.((xz)(yz))), \\ (\lambda y.(\lambda z.((xz)(yz)))), \quad (\lambda x.(\lambda y.(\lambda z.((xz)(yz))))))$$

Conventions

- Conventions for removing parentheses:
 1. Omit outermost parentheses, e.g. MN instead of (MN)
 2. Applications are left-associative, omit parentheses when not necessary, e.g. MNP means $(MN)P$
 3. Body of abstraction extends to right as far as possible, e.g. $\lambda x.MN$ means $\lambda x.(MN)$
 4. Nested abstractions can be contracted, e.g. $\lambda xy.M$ means $\lambda x.\lambda y.M$

Ex: Write the following with as few parentheses as possible:

$$(\lambda x.(\lambda y.(\lambda z.((xz)(yz)))))) \Rightarrow \lambda xyz.xz(yz)$$

Free and bound variables

- In the term $\lambda x.M$
 - λ is said to bind x in M
 - λx is called a binder
 - x is a bound variable
- In the term $\lambda x.xy$
 - x is a bound variable
 - y is a free variable
- In the term $(\lambda x.xy)(\lambda y.yz)$
 - x is a bound variable
 - z is a free variable
 - y has a free and a bound occurrence
 - Set of free variables $FV=\{y,z\}$

Set of free variables

- $FV(M)$: the set of free variables of a term M

$$- FV(x) = \{x\},$$

$$- FV(\lambda x.M) = FV(M) - \{x\}$$

$$- FV(MN) = FV(M) \cup FV(N),$$

- Set of free variables in term M defined as

$$\lambda xy.((\lambda z.\lambda v.z(zv))(xy)(zu))$$

is :

$$\begin{aligned} FV(M) &= FV((\lambda z.\lambda v.z(zv))(xy)(zu)) - \{x, y\} \\ &= (FV(\lambda z.\lambda v.z(zv)) \cup FV(xy) \cup FV(zu)) - \{x, y\} \\ &= ((\{z, v\} - \{z, v\}) \cup \{x, y\} \cup \{z, u\}) - \{x, y\} \\ &= \{z, u\} \end{aligned}$$

α -equivalence

- $\lambda x.x$ is the same as $\lambda y.y$ (both are identity function)
- $\lambda x.x^2$ is the same as $\lambda z.z^2$
- Renaming bound variables does not change the abstraction
- This is called α -equivalence of lambda terms and is denoted as

$$\lambda x.M =_{\alpha} \lambda y.(M \{x \setminus y\})$$

- Where $M\{x \setminus y\}$ denotes renaming every occurrence of x in M to y (assuming y does not already occur in M)
 - Note x is a bound variable in this definition

Substitution

- Substitution is defined for free variables, substituting a variable with a term.
 - $(\lambda x.xy)[y := M] = \lambda x.xM$
 - $(\lambda x.xy)[y := (uv)] = \lambda x.x(uv)$
- Substitution must be defined to avoid capture
 - $(\lambda x.xy)[y := x] \neq \lambda x.xx$
 - $(\lambda x.xy)[y := x] = (\lambda x'.x'y)[y := x] = \lambda x'.x'x$

 - $(\lambda x.yx)[y := (\lambda z.xz)] \neq \lambda x.(\lambda z.xz)x$
 - $(\lambda x.yx)[y := (\lambda z.xz)] = \lambda x'.(\lambda z.xz)x'$

Substitution (cont.)

- Definition:

$$x[x := N] \equiv N$$

$$y[x := N] \equiv y \quad \text{if } x \neq y$$

$$(MP)[x := N] \equiv (M[x := N])(P[x := N])$$

$$(\lambda x.M)[x := N] \equiv \lambda x.M$$

$$(\lambda y.M)[x := N] \equiv \lambda y.(M[x := N]) \quad \text{if } x \neq y \text{ and } y \notin FV(N)$$

$$\rightarrow (\lambda y.M)[x := N] \equiv \lambda y'.(M\{y \setminus y'\}[x := N]) \quad \text{if } x \neq y, y \in FV(N), \text{ and } y' \text{ fresh}$$

Capture case!

Bound variable y is **renamed** to y' to
avoid capture of free variable y in N

β -reduction

- β -reduction: the process of evaluating a lambda term by giving value to arguments

For example:

- $(\lambda x.x^2)(5) \rightarrow_{\beta} 25$
- $(\lambda x.y)(z) \rightarrow_{\beta} y$

- Definition

- β -redex: A term of the form $(\lambda x.M)N$ (a lambda abstraction applied to another term)
- It reduces to $M[x:=N]$
- The result is called a reduct
- β -reduction is applied recursively until there is no more redexes left to reduce
- A lambda term without any β -redexes is said to be in β -normal form

β -reduction – more examples

- $(\lambda x.y)(\lambda z.zz) \rightarrow_{\beta} y[x:=(\lambda z.zz)] = y$
- $(\lambda x.y)(\lambda w.w) \rightarrow_{\beta} y[x:=(\lambda w.w)] = y$
- $(\lambda w.w)(\lambda w.w) \rightarrow_{\beta} w[w:=(\lambda w.w)] = (\lambda w.w)$
- $(\lambda x.y)((\lambda z.zz)(\lambda w.w))$
 $\rightarrow_{\beta} (\lambda x.y) (zz [z:=(\lambda w.w)]) \rightarrow_{\beta} (\lambda x.y) ((\lambda w.w) (\lambda w.w))$
 $\rightarrow_{\beta} (\lambda x.y) (\lambda w.w) \rightarrow_{\beta} (y [x:= (\lambda w.w)]) \rightarrow_{\beta} y$
- Or $(\lambda x.y)((\lambda z.zz)(\lambda w.w))$
 $\rightarrow_{\beta} y [x:= ((\lambda z.zz)(\lambda w.w))] \rightarrow_{\beta} y$

Why Lambda Calculus?!

Popular question in 1930's:

“What does it mean for a function f to be computable?”

- Intuitive computability: A pencil-and-paper method to allow a trained person to calculate $f(n)$ for any given n ?
- 1. **Turing**: A function is computable if and only if it can be computed by the Turing machine.
- 2. **Gödel**: A function is computable if and only if it is general recursive.
- 3. **Church**: A function is computable if it can be written as a lambda term.
- It has been proven that all three models are equivalent.
- Are they equivalent to ‘intuitive computability’? Cannot be answered!

Lambda Calculus as a Programming Language

- Lambda calculus
 - It can be used to encode programs AND data, such as Booleans and natural numbers
 - It is the simplest possible programming language that is Turing complete
 - ‘Pure LISP’ is equivalent to Lambda Calculus
 - ‘LISP’ is Lambda calculus, plus some additional features such as data types, input/output, etc